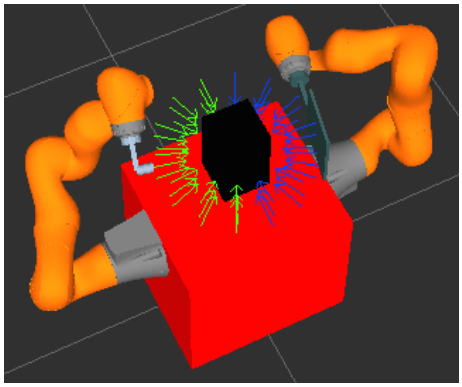
Technische Universität München
Fakultät für Informatik
Dr. Tobias Lasser

WS 2016
Exercise Sheet 6
December 6, 2016

Exercises in Image Reconstruction and Visualization using C++

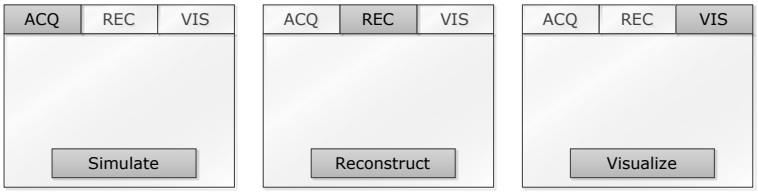
This exercise represents the first part of your project. The goal of this part is to implement simulated image acquisition using a flexible setup of X-ray source and detector. An example setup is shown in the following figure, where two robotic arms hold the X-ray source and the detector (poses of the source are marked in green, poses of the detector in blue):



Exercise 1 Setting up the project

- Make sure you have access to your team's assigned git repository on `gitlab.lrz.de`. Use this git repository for your work on part 2.
- Constantly keep making sure that our gitlabCI runners build your code successfully. This is a hard requirement for passing! Commit and push early and often.

  (Note: we are not testing your functionality automatically like in part 1, this is now purely your own responsibility!)
- You will need to include Eigen and Qt5 as external dependencies (via CMake, as done in assignment 5).
- Set up a basic Qt user interface as sketched out below. Each part of the project (acquisition, reconstruction, visualization) will get its own widget, for example controlled via tabs as in the sketch. Each part will also have a central push button (simulate, reconstruct, visualize) to execute the tasks for each part of the project. You will fill the rest of the widget with relevant information as you think best.



Exercise 2 The imaging setup and the volume of interest

First we need to describe our imaging setup. We will have a flexible X-ray source, for example mounted on a robotic arm, as in the picture above. It will send (simulated) X-rays through a volume of interest, indicated as the black box in the picture above. A flexible X-ray detector, for example mounted on another robotic arm, as in the picture above, will record the (simulated) X-rays after they passed through the volume of interest.

We set the origin of our world coordinate system at the lower left front corner of the black box (our volume of interest), the units we use are in meters. The black box has dimensions $0.15m \times 0.15m \times 0.25m$. We discretize the black box using voxels of size $0.015m \times 0.015m \times 0.025m$, such that we get $10 \times 10 \times 10$ voxels. This resolution is very coarse in order to keep computation times low, in practice you would use finer resolutions.

We will provide you with the content of this volume of interest in the .edf file format. Some example volumes in that format for you to play around with are available on the course website, along with C++ code to handle the file format. Your final program will take one "mystery" volume in this .edf format as a command line argument, i.e. `./<yourprogram> mystery.edf`.

- Familiarize yourself with the .edf file format.
- Implement reading and writing of these volumes using the provided code. You will need to implement a suitable `Volume` class for this, you can use the provided header `volume.h` as a starting point.

Hint: Volumes are usually stored in a linearized format. Here we linearize first in x-, then y-, then z-direction. That means in order to convert from 3D volume coordinates (x,y,z) to a linearized 1D array index, we compute

$$index = x + y * NX + z * NX * NY,$$

where NX, NY, NZ are the number of voxels in the volume in x-, y- and z-direction. Converting from index to volume coordinates works accordingly.

Exercise 3 The geometry and acquisition poses

The position and orientation of the X-ray source and the detector in relation to the volume of interest define the geometry of one *acquisition pose*. One full image acquisition for tomographic reconstruction then consists of several of these acquisition poses.

We will be using a detector with dimensions $0.2m \times 0.2m$ and pixels of size $0.04m \times 0.04m$, that is the detector has 5×5 pixels. Again, the resolution is very coarse to keep computation times low, in practice resolutions will be finer. As a simplification we will assume that we record X-rays at the center of each detector pixel.

- Implement classes that represent an acquisition pose. Ensure that source and detector are always oriented towards each other!
- Pick a few example acquisition poses to play around with. (You will choose a suitable trajectory of acquisition poses later on.)
- Visualize your acquisition poses somehow in the Qt GUI. For example, a simple 2D projection of the scene suffices.
- Provide an interface to get the ray (i.e. the origin and the direction of the ray) from the X-ray source to the center of a detector pixel for a specific acquisition pose and a specific pixel on the detector.

Exercise 4 Ray-tracing

Implement a ray-tracer based on the rays from the previous exercise. This ray-tracer should compute the entry point to the volume of interest along the ray, and from there it should step through the volume performing some action on each voxel that is hit by the ray (see next exercise for the action).

- Implement a ray/volume intersection method to compute the entry point of the ray.
- Implement a voxel traversal method to step through the volume along the ray.

See the provided slides and literature for hints on how to do this. As this ray-tracer will be called very very often, it has to be implemented efficiently.

Exercise 5 Forward projection and simulation of X-rays

Now we want to simulate the X-ray images that your images setup would acquire of the object in the volume we provide you as an .edf. For this we need to implement the forward projection operator as described in the provided slides.

- Iterate over all the rays in your trajectory (i.e. over all acquisition poses and all detector pixels), call your ray-tracer from the previous exercise for each ray.
- Compute the forward projection of the volume of interest by using the results of the ray-tracer.
- Implement the previous steps as one forward projection operator. Apply this operator to one of the volumes read in from exercise 1. The result will be the simulated X-ray images that we will use later on for reconstruction.
- Visualize the simulated images somehow in the Qt GUI.
- Make sure that your simulated X-ray images make sense.